useLayoutEffect

- It is a alternative of useEffect hook but it **preventing any flickering or visual inconsistencies**.
- The code inside useLayoutEffect and all the state updates scheduled, **block the browser from repainting the screen.** When used excessively, this makes your **app slow**. When possible, **prefer useEffect**.
- useLayoutEffect hook takes a function called **side effect** as its first argument and **an array of dependencies** as second argument. The first argument, effect, either returns a **cleanup function or undefined.**
- useLayoutEffect is usually used **together with the useRef** hook, which will allow you to get a reference to any DOM element that you can use to read layout information.

When to choose useLayoutEffect over useEffect hook: **useEffect hook is called after the screen is painted**. Therefore **mutating the DOM again immediately** after the screen has been painted, **will cause a flickering effect** if the mutation is visible to the client. **useLayoutEffect hook on the other hand is called before the screen is painted but after the DOM has been mutated.**

useLayoutEffect	**useEffect**
Run synchronously which means it runs immediately	Run Asynchronously which means it runs not immediately
Before browser rendering	After browser rendering
Potentially blocking, may delay rendering	Non-blocking, does not delay rendering

Pitfalls of using the useLayoutEffect hook:

- It is that it can **hurt app performance**.
- **No support for Server-Side Rendering (SSR):** Because SSR requires asynchronous rendering to avoid blocking the server thread.

What's side effect?: You might need to carry out some tasks or operations when working on a React project that falls outside the render cycle of the component. These are known as "Side Effects". A side effect is anything that happens within your application that is not (at least not directly) related to UI rendering. For example, send HTTP requests to servers, store data in the browser's memory, and set time functions. There are no UI changes in these scenarios. In other words, React will not re-render your component for these scenarios.

Usage:

1.Measuring layout before the browser repaints the screen: Imagine a tooltip that appears next to some element on hover. If there's enough space, the tooltip should appear above the element, but if it doesn't fit, it should appear below. In order to render the tooltip at the right final position, you need to know its height (i.e. whether it fits at the top).

2. Auto-focus Input Field:Auto-focus Input Field:

```
import React, { useRef, useLayoutEffect } from "react";

const AutoFocusInput = () => {
  const inputRef = useRef(null);

  useLayoutEffect(() => {
    inputRef.current.focus();
  }, []);

  return <input ref={inputRef} />;
};
```

3.Animating Elements:

```jsx
import React, { useRef, useLayoutEffect } from "react";

const AnimatingElements = () => {
  const elementRef = useRef(null);

  useLayoutEffect(() => {
    const element = elementRef.current;

    // Animate the element's opacity on mount
    element.style.opacity = 0;
    setTimeout(() => {
      element.style.opacity = 1;
    }, 1000);

    return () => {
      // Clean up animation when the component unmounts
      element.style.opacity = 0;
    };
  }, []);

  return <div ref={elementRef}>Animate me!</div>;
};
```

4.Adding Smooth scrolling:

```jsx
import React, { useRef, useLayoutEffect } from "react";

const SmoothScrolling = () => {
  const containerRef = useRef(null);

  useLayoutEffect(() => {
    const container = containerRef.current;

    const handleScroll = () => {
      // Smoothly scroll to the top of the container
      container.scrollTo({
        top: 0,
        behavior: "smooth",
      });
    };

    // Scroll to the top when the component is mounted
    handleScroll();

    // Add event listener to scroll to the top on subsequent scrolls
    window.addEventListener("scroll", handleScroll);

    return () => {
      window.removeEventListener("scroll", handleScroll);
    };
  }, []);

  return <div ref={containerRef}>{/* Your Content */}</div>;
};
```